COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS Versions 5 & 6

June 1994
Vol. 5 No. 6

VERSION 6.2

Copying over files in DOS 6.2

In the days before DOS 6.2, you faced a degree of risk when you used the COPY, XCOPY, and MOVE commands. If you copied a file to a directory that contained a file with the same name, the new file copied over the existing file without letting you know. Many users complained about this situation, so Microsoft introduced in DOS 6.2 a safety feature for the three copy commands. If you're about to overwrite a file, DOS presents the prompt

```
Overwrite filespec (Yes/No/All)?
```

which asks you to verify whether you want to copy over the existing file.

This Copy Overwrite Protection feature keeps you from accidentally overwriting files. However, if you intend to replace old information with updated files, this feature can be a hassle.

Fortunately, Microsoft gave the copy commands some new switches that let you turn the *Overwrite...* prompt off and on. DOS 6.2 includes a new environment variable, called COPYCMD, that lets you specify whether you want the COPY, XCOPY, and MOVE commands to ask you for confirmation before overwriting a file.

In this article, we'll show you how to use the new command switches to suppress the *Overwrite...* prompt when you use a COPY, XCOPY, or MOVE command. Then we'll show you how to globally set the switches in the COPYCMD environment variable and how to override those global settings. Finally, we'll tell you how the new feature affects batch files.

Suppressing the prompt at the command line

You can suppress the *Overwrite...* prompt when you invoke any of the copy commands in DOS 6.2. All you do is add the switch /Y at the end of the COPY and XCOPY commands, like this

```
copy filespec destination /y
```

where *filespec* is the path and name of the file(s) to copy and *destination* is the drive, directory, and/or filename to which you're copying *filespec*. Suppose you regularly update a group of Excel spreadsheets. You keep copies of all files in a pending directory and copy only the ones you want to update to a working directory before you open them in Excel. You can do this with the command

```
C:\EXCEL>copy pend\hw9406.* work /y
```

With the MOVE command, you place the /Y switch after the command name:

```
move /y filespec destination
```

Let's say you've updated the spreadsheets and you now want to move them from your working subdirectory to your pending subdirectory, which holds copies of all working files. You know what you're doing, and you want to avoid having to answer the *Overwrite...* prompt. This command will do the job:

```
C:\EXCEL>move /y work\hw9406.* pend
```

When you press [Enter], DOS doesn't ask you whether you want to overwrite the existing files that have the same name. Instead, DOS lists the files it's

IN THIS ISSUE

moving and confirms that the files moved successfully, like this:

```
c:\excel\work\hw9406.xls => c:;\excel\pend\hw9406.xls [ok]
c:\excel\work\hw9406.xlm => c:;\excel\pend\hw9406.xlm [ok]
c:\excel\work\hw9406.bak => c:;\excel\pend\hw9406.bak [ok]
c:\excel\work\hw9406.xlc => c:;\excel\pend\hw9406.xlc [ok]
```

If you generally like the security of the *Overwrite...* prompt, you should use this method to suppress it temporarily in those cases when you absolutely know what you're doing. However, if you want to suppress the prompt on a more permanent basis, you should set the COPYCMD environment variable. We'll show you how to do that next.

Setting the COPYCMD environment variable

The COPYCMD variable, new to DOS 6.2, controls the operation of the COPY, XCOPY, and MOVE commands. You set COPYCMD in order to globally assign switches to these three commands.

Environment variables store information during the current DOS session. You establish their values by issuing the SET command. To establish COPYCMD as an environment variable, you use the syntax

```
set copycmd=/switch
```

The COPYCMD variable has two possible switches: /Y and /-Y. The /Y switch forces the COPY, XCOPY, and MOVE commands to overwrite without prompting you, whereas the /-Y switch restores the prompt. For instance, to turn off the *Overwrite...* prompt for the current DOS session, you enter the command

```
C:\>set copycmd=/y
```

Now when you issue a COPY, XCOPY, or MOVE command, DOS will execute the command without telling you that a file already exists and asking if you want to overwrite it. Conversely, to restore the prompt you enter the command

```
C:\>set copycmd=/-y
```

Since environment variables exist only during a single DOS session, you must reset COPYCMD each time you reboot your computer. If you want to set the COPYCMD variable permanently, you put the environment setting in your CONFIG.SYS file. Let's do that next.

Setting COPYCMD permanently

You can add the SET command anywhere in your CONFIG.SYS file to set the COPYCMD variable each

INSIDE DOS

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices:
Domestic: $49/yr ($6.00 each)
Outside US: $69/yr ($8.50 each)

Phone:
Toll free: (800) 223-8720
Local: (502) 491-1900
Customer Relations Fax: (502) 491-8050
Editorial Department Fax: (502) 491-3433

Address:
You may address tips, special requests, and other correspondence to

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster:
Second class postage is pending in Louisville, KY. Send address changes to

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Authorized Canada Post International Publications Mail (Canadian Distribution) Sales Agreement #XXXXXX CANADA GST #123669673. Send returns to Canadian Direct Mailing Sys. Ltd., 920 Mercer Street, Windsor, Ontario, N9A 7C2. Printed in the USA.

time you boot or reboot your computer. You could edit the CONFIG.SYS file in the DOS Editor or another ASCII text editor. However, you can add the new line to the existing file by using the ECHO command and the >> append symbol. At the DOS prompt, simply type the command

```
C:\>echo SET COPYCMD /Y >> config.sys
```

and press [Enter].

When you reboot your computer, the COPYCMD environment variable will control the behavior of the COPY, XCOPY, and MOVE commands. As a result, DOS won't display the *Overwrite...* prompt before it overwrites any existing file. At least, it won't unless you tell it to restore the prompt.

Restoring the prompt

Even if you use the COPYCMD variable to establish default settings for COPY, XCOPY, and MOVE, you might occasionally want to restore the *Overwrite...* prompt for a single command or DOS session. Fortunately, reversing COPYCMD's setting is as easy as setting it.

The key to restoring the prompt is the /-Y switch. If you want to restore the prompt for a single command, you use the /-Y switch when you invoke the command. For example, if you set the COPYCMD variable to suppress the prompt, you restore the prompt for a single XCOPY command like this:

```
C:\>xcopy hw 9406.* b: /-y
```

If, on the other hand, you want to restore the prompt for several commands or for the rest of the DOS session, you use this command:

```
C:\>set copycmd=/-y
```

That's all there is to it. To suppress the prompt again, just replace the /-Y switch in the above command with the /Y switch.

Of course, after a few rounds of setting and resetting the COPYCMD variable, you might forget whether the prompt is turned on or off. If that should happen, you can check the status of the COPYCMD variable the same way you check the status of any other environment variable—simply issue the SET command at the DOS prompt. When you do, you'll see the status of all your environment variables, which might look something like this:

```
C:\>set
COMSPEC=C:\DOS\COMMAND.COM
PROMPT=$p$g
MOUSE=C:\MOUSE
PATH=c:\DOS;c:\;c:\WINDOWS;c:\DOCS;c:\BATCH;c:\mouse
TEMP=C:\DOS
COPYCMD=/Y
```

A note about batch files

As you can imagine, any batch file that uses the COPY, XCOPY, or MOVE command would be seriously impaired if it had to stop and prompt the user to overwrite existing files. Microsoft anticipated that this would be a problem and disabled the *Overwrite...* prompt within batch files. The commands work the way they always have in batch files: They automatically overwrite any file that has the same name as the one being copied.

Conclusion

DOS 6.2 introduced the Copy Overwrite Protection feature to give users added safety when using the COPY, XCOPY, and MOVE commands. In this article, we showed you how to disable this feature using commands at the DOS prompt and in the CONFIG.SYS file. Also, we showed you how to set the COPYCMD environment variable to set the default action for the COPY, XCOPY, and MOVE commands. ◆

VAN WOLVERTON

VERSION 5.0 & 6.x

Charting a PATH to commands

One of the most disconcerting messages a new DOS user encounters is *Bad command or file name*. This curt dismissal is frustrating, but you quickly learn that its most common causes are trivial—either you misspelled the name of a command or DOS can't find a command file because the directory that contains the DOS files isn't in the command path. On the other hand, the message may be your first warning that something more serious has happened (such as all your DOS files have been erased) but those occasions are blessedly rare.

The command path, which you define with the PATH command, is a fairly simple concept: It tells DOS which directory or directories to search for a command file if the file isn't in the current directory. The DOS file structure is hierarchical, based on a layered system of

directories that contain files or subdirectories, which in turn contain files or subdirectories, which in turn.... You know the routine. The command path makes this file structure much easier to use than it would be otherwise.

Without a command path, you can't use a command that isn't in the current directory unless you precede the command name with the path to the command. For example, if you're in the DOS directory, you'd have to use the command

```
C:\DOS>\batch\size
```

to run the SIZE.BAT batch file, which is stored in the BATCH directory. This rule applies to batch files, too—if a batch file calls a command file, either the command file must be in the same directory as the batch file or you must list the full path to the command file in the statement that calls the command file.

The command path lets you ignore all these restrictions. DOS stores the definition of the command path in the environment as the environment string that begins *PATH=*. By including in AUTOEXEC.BAT a PATH command that specifies the names of directories that contain the commands you most commonly use, you can do your work without paying much attention to the current directory.

The PATH command

The PATH command uses straightforward syntax:

- If you type *PATH* followed by an equal sign and a series of directory names separated by semicolons, DOS defines the command path to consist of these directories.
- If you type *PATH*, DOS displays the environment string that defines the command path (or, if you haven't defined a command path, DOS displays the message *No path*).

For example, the following PATH command tells DOS to search the directory that contains the DOS files (C:\DOS) and the directory that contains your batch files (C:\BATCH):

```
path=c:\dos;c:\batch
```

If you type *path* at this point, DOS displays this statement:

```
PATH=C:\DOS;C:\BATCH
```

The only restriction on the length of the command path is the length of a DOS command line—127 characters. Don't try to cram more directories into the path statement by omitting the drive letter and colon: The command path mechanism won't work properly if you don't include them (and besides, you shouldn't need that many directories in your command path, anyway).

The command path isn't complicated and operates invisibly if you've included all the directories that you commonly use. However, there are still a few things you can do to keep your system running smoothly.

Speeding the search for a command

When you type something at the command prompt and press [Enter], DOS assumes you've typed the name of a command. It checks to see if the first word you typed matches the name of a file in the current directory whose extension is COM, EXE, or BAT, in that order. If it doesn't find such a file, it checks the environment to see if you've defined a command path with the PATH command. If you have, it looks for the command file in each directory in the command path, starting with the first directory that follows the equal sign.

Understanding this search sequence reveals one of the ways you can help DOS operate more efficiently. Because you use some types of commands more frequently than others, you should put the directories that contain those commands at the beginning of the PATH command in AUTOEXEC.BAT. You know your file structure and pattern of command use better than anyone, so you'll have to pick the sequence that best matches the way you use your computer.

For example, it's a pretty safe bet that you use DOS commands more than any other collection of commands, so you really can't go wrong putting the DOS directory first:

```
path=c:\dos
```

If you spend most of your time with one or two application programs—for example, Microsoft Word and Generic CADD—put those directories next:

```
path=c:\dos;c:\word;c:\cadd6
```

Next come the directories that contain your batch files, utility programs, games, application programs you use less frequently, and other directories that contain programs you use. It isn't necessary to include every directory that contains a program, however. For example, if you use a batch file to run an application program, you can set up the batch file so it changes to the directory that contains the program. Thus, only the directory that contains the batch file must be in the command path.

Beware the helpful installation program

All too often when you install a new application program or game, the installation program adds the directory in which it installed the program to your PATH command

in AUTOEXEC.BAT. If you're lucky, it *offers* to make this change, giving you a chance to decide whether you really need the addition to your command path. If you're not so lucky, it just adds the directory to your command path without letting you intervene. Your command path grows as you install more programs, often getting bloated with directories you don't even need.

And if you're *really* unlucky, the installation program simply appends to your AUTOEXEC.BAT a new PATH command that specifies only the directory that contains the new program files. This new command path, unfortunately, replaces any previous command path you may have defined. This new path has the virtue of being quite short, but it makes your system supremely inconvenient to use. Happily, most of the people who write installation programs are smart enough to avoid creating this problem, but it still happens once in a while.

If you're given a choice, don't let a program you're installing alter your AUTOEXEC.BAT file. Usually, an installation program wants access to AUTOEXEC.BAT only to change your PATH statement—so you're better off refusing to let it change the file at all. After you've installed the program, use EDIT to check AUTOEXEC.BAT and make sure the installation program didn't make any changes without telling you. If it added a directory to the PATH command, delete that directory. You can always put the directory back in if the program refuses to work correctly.

Using batch files to manage your command path

You can create a couple of batch files to let you manage your command path a bit more easily. One, ADDPATH.BAT, lets you temporarily add a directory to your command path. The other, SHOWPATH.BAT, simply displays the directories in your command path in a more readable form than DOS normally does.

Adding a directory to the command path

ADDPATH.BAT is a short batch file that takes advantage of the fact that you can refer to the value of an environment variable in a batch file by enclosing the name of the variable in percent signs. The batch file contains only two lines:

```
@echo off
path=%path%;%1
```

The first line simply tells DOS not to echo commands in the batch file before it executes them. All the work in the batch file is really done in the PATH command. It defines the command path as the value of the existing PATH environment variable—in other words, the current command path—plus a semicolon and %1, the parameter you type with the batch command.

To add the directory C:\TELECOM\PCPLUS to your command path, you type

```
C:\>addpath c:\telecom\pcplus
```

This addition to your command path is effective only until you restart your system. To make the change permanent, open AUTOEXEC.BAT in the DOS Editor and add the directory to the PATH command.

Displaying your command path

As we showed you, when you type *path* by itself, DOS displays the PATH environment variable (or the message *No path*). This path statement isn't particularly easy to read, especially if there are several directories in the command path. The batch file SHOWPATH.BAT takes advantage of another DOS capability available only with the FOR command in batch files: If you include as the set variable an environment variable whose value consists of a series of individual values separated by semicolons, DOS breaks the compound value apart into the individual values.

All the work of the SHOWPATH.BAT file, like ADDPATH.BAT, is really done by the FOR command. The batch file consists of five lines:

```
@echo off
echo.
echo Command Path Directories
for %%p in (%PATH%) do echo %%p
echo.
```

The ECHO commands turn off command echoing, display a blank line, and display a heading for the command output. The FOR command sends each value in the PATH environment variable to the ECHO command that follows DO. The final ECHO command displays another blank line to separate the command output from the following command prompt.

Suppose your command path consisted of five directories: C:\DOS, C:\WORD, C:\CADD6, C:\BATCH, and C:\UTILS. The normal output of the PATH command with no parameter would be

```
PATH=C:\DOS;C:\WORD;C:\CADD6;C:\BATCH;C:\UTILS
```

But type *showpath*, and you'll see output that's much easier to read:

```
Command Path Directories

C:\DOS
C:\WORD
C:\CADD6
C:\BATCH
C:\UTILS
```

These batch files aren't earthshaking, but each little personal touch you add makes your computer more pleasant to use. These batch files can help ensure that your command path covers the programs you use. ◆

Contributing editor Van Wolverton is the author of the best-selling books Running MS-DOS *and* Supercharging MS-DOS. *Van, who has worked for IBM and Intel, lives in Alberton, Montana.*

BATCH FILE COMMANDS

VERSION 5.0 & 6.x

Using ERRORLEVEL to make decisions in a batch file

Many of the batch files we show you in *Inside DOS* use the cryptic IF ERRORLEVEL command. This command lets you add decision-making capabilities to your batch files. By testing ERRORLEVEL, you can tell the batch file to execute one action for a certain condition and another action for a different condition.

Most books and DOS documentation skim over the subject, giving the impression that "regular DOS folk" don't really need to know about ERRORLEVEL. However, ERRORLEVEL shouldn't be reserved just for programmers. Once you understand how to use it, you can add a lot of power to your batch files. In this article, we'll tell you what ERRORLEVEL is, where it originates, and how to use it effectively in a batch file.

What is ERRORLEVEL and where does it originate?

You use the IF command to test for a variety of conditions, such as whether a file exists or whether a parameter matches a text string. Another condition you can test with the IF command is the content of the ERRORLEVEL variable. The ERRORLEVEL variable holds the most recently created exit code. Exit codes are messages created by a few DOS commands and by some EXE and COM programs when the command or program ends. The ERRORLEVEL variable can hold only a single byte of information, so an exit code is actually a one-byte code that represents a specific message. Because *ERRORLEVEL* and *exit code* are so closely related, we often use the terms interchangeably.

Only a few DOS commands produce exit codes. Those codes usually tell whether the command ran successfully or couldn't run (it found a system error or the user canceled it). For example, the XCOPY command produces an exit code when it finishes executing. You could test this exit code in a batch file and branch to a message that tells you to replace the target disk if the disk is suspect (exit code 5), to exit the batch file if you pressed [Ctrl]C (exit code 2), or to loop back and copy more files if the first XCOPY command executed successfully (exit code 0). Unfortunately, the COPY command (XCOPY's sibling) doesn't produce exit codes, so you have to write more creative batch file commands to test whether it executed successfully.

The name *ERRORLEVEL* is misleading because for most programs, the variable usually doesn't indicate an error. For example, the DOS 6 CHOICE command creates exit codes you can test in a batch file. You list the allowable keys in your CHOICE statement, and DOS assigns each of the keys a sequential exit code from 1 to *n*. When you run the batch file, CHOICE waits for you to press one of the keys and assigns that key's exit code to the ERRORLEVEL variable. The batch file won't continue until you press one of the allowed keys.

Before the CHOICE command, we relied on public domain DEBUG programs with names like GETKEY.COM and ASK.COM for a way to solicit input from the user. Most of these programs pause the batch file, wait for you to press a key, and assign the ASCII value (from 0 to 255) of that keypress to the ERRORLEVEL variable. You display a message that lists the allowable keys and set up IF ERRORLEVEL statements to test for all the possible keypresses. If you include a message like *Press any other key to quit*, you have to test for all 255 possible ASCII values.

Now that we've looked at what ERRORLEVEL is and where exit codes originate, let's look at how you can test the ERRORLEVEL variable in batch files.

Testing ERRORLEVEL

You test the exit code stored in the ERRORLEVEL variable by using an IF statement in the form

```
if errorlevel n command
```

When a batch file evaluates this command, it executes the *command* if the exit code stored in ERRORLEVEL is equal to or greater than *n*. For example, if you use the command

```
if errorlevel 1 goto :SUB1
```

in a batch file, control branches to :SUB1 when ERRORLEVEL is equal to or greater than 1.

You can also use an IF NOT statement to test ERRORLEVEL. The statement

```
if not errorlevel n command
```

means to execute the *command* if ERRORLEVEL is less than *n*. If you use the command

```
if not errorlevel 3 goto :SUB3
```

in a batch file, control branches to :SUB3 when ERRORLEVEL is 0, 1, or 2.

You rarely test only a single ERRORLEVEL variable in a batch file. You usually want to test several exit code values so you can send the batch file to different branches, depending on specific ERRORLEVEL values. However, because DOS evaluates an IF ERRORLEVEL statement as *greater than or equal to* a specific value, you have to force DOS to evaluate as being *equal to* the ERRORLEVEL value. Fortunately, there are a couple of ways to ensure that DOS interprets your IF statements correctly.

One way to force DOS to evaluate exact numbers in your IF ERRORLEVEL statements is to combine an IF and IF NOT clause into a single statement. You can use a statement in the form

```
if errorlevel n if not errorlevel n+1 command
```

and the batch file will execute *command* if the exit code is greater than or equal to *n* and is less than *n+1*—in other words, if the exit code is exactly *n*.

There's an even easier way to test for exact numbers. You simply place your IF ERRORLEVEL statements in order from highest number to lowest number. We'll show you how to do that next.

Combining ERRORLEVEL tests in a batch file

Knowing how to test ERRORLEVEL is only one piece of the puzzle. You also need to know how to combine IF ERRORLEVEL statements in a batch file so DOS will test them correctly and branch to the appropriate label. Fortunately, testing ERRORLEVEL is easy when you have a defined group of exit codes. Now we'll demonstrate how to arrange a block of IF ERRORLEVEL statements.

As we mentioned, several DOS commands produce exit codes when they finish running. For example, FORMAT's exit codes are

0	Successful format
3	User pressed [Ctrl]C to cancel format
4	Fatal error, format incomplete
5	User pressed N at *Proceed with Format (Y/N)?* prompt

When you want to test each exit code, you check ERRORLEVEL's values in reverse order. When we use the series of statements

```
if errorlevel 5 goto :NOPROCEED
if errorlevel 4 goto :FATAL
if errorlevel 3 goto :CANCEL
if errorlevel 0 goto :SUCCESS
```

the batch file first checks ERRORLEVEL for a value of 5 or higher. If the batch file doesn't find a value of 5 or higher, it moves to the second statement and checks for a value of 4 or higher. Since we already checked for a value of 5, which is higher than 4, the second statement checks for *only* the value 4. In effect, by placing the statements in reverse order, you force the batch file to check for *exact* ERRORLEVEL values instead of values that are equal to or greater than the value you want to test.

You also can test ERRORLEVEL for a range of exit code values. Continuing with our example, you may want your batch file to branch to one of two places—to the label :SUCCESS if the format operation completed or to :NOSUCCESS if it failed or if you canceled it. In this case, you can boil down the ERRORLEVEL test to two statements:

```
if errorlevel 3 goto :NOSUCCESS
if errorlevel 0 goto :SUCCESS
```

The first statement sends the batch file to :NOSUCCESS if ERRORLEVEL contains the value 3, 4, or 5, and the second statement sends the batch file to :SUCCESS if ERRORLEVEL contains the value 0, 1, or 2. (Of course, 1 and 2 aren't valid FORMAT exit codes.)

You use the GOTO command with all your IF ERRORLEVEL statements to force the batch file to branch out of the set of IF ERRORLEVEL statements. You do this to ensure that once the batch file finds a statement that evaluates to True, it doesn't test the next statement, which may also be true. This point brings us to another vital part of the ERRORLEVEL discussion: program flow.

Program flow: putting the statements in context

All batch files are linear—you type commands one after the other on separate lines. Most batch files have fairly linear logic, too—the batch file executes the first command and then the second command, followed by the third command, and so forth, until it reaches the final command. All the commands in the batch file combine to perform a specific task.

A batch file loses its linear logic when you use IF commands to test conditions, since you want the batch file to perform different tasks under different circumstances. For example, you want the batch file to perform task X only if condition A exists, task Y only if condition B exists, and so on.

However, even though these batch files have nonlinear logic, you must type them in a linear fashion—

one command after the other. You control how the batch file executes the commands that perform the correct task by arranging the commands in the appropriate order. In this situation, you use the GOTO command to make the batch file jump over a group of commands to a branch label.

This type of batch file is easiest to understand if we arrange the commands in functional blocks. First, we set up the decision block—in a batch file that tests the ERRORLEVEL variable, that's a block of IF ERRORLEVEL statements. Each statement sets up a condition and a preview of its consequences. For example, the block we showed you earlier

```
if errorlevel 5 goto :NOPROCEED
if errorlevel 4 goto :FATAL
if errorlevel 3 goto :CANCEL
if errorlevel 0 goto :SUCCESS
```

sets up four conditions and their corresponding consequences. Each consequence is a separate action, so we must set up separate action blocks for each one.

Obviously, we need four action blocks:

```
:NOPROCEED
some command
goto :END

:FATAL
some other command
goto :END

:CANCEL
yet another command
goto :END

:SUCCESS
the last of these commands
goto :END

:END
```

Each labeled block executes an appropriate command or series of commands. We could place the blocks in any order; however, for clarity, we put them in the same order as the IF ERRORLEVEL statements.

We ended each action block with the command

```
goto :END
```

which forces the batch file to skip over the commands it doesn't need to execute. After the batch file executes the appropriate action, it branches to the label :END and quits.

Of course, your batch file may contain additional blocks for error messages or alternative commands. If so, you can place the extra blocks before or after the action blocks. Just be sure that any new commands you add to the batch file send control to the appropriate branch.

Notes

The DOS commands that create exit codes when they finish running are BACKUP, DEFRAG, DISKCOMP, DISKCOPY, FORMAT, GRAFTABL, KEYB, REPLACE, RESTORE, SETVER, and XCOPY. You can find exit code values for these commands in your DOS documentation. In DOS 5, check your User's Guide, and in DOS 6, check the Notes section in the online help for the command in question.

Conclusion

Some DOS commands and programs produce exit codes when they run. You can add power to your batch files by testing the exit code, which is stored in the ERRORLEVEL variable. Depending on the value in ERRORLEVEL, you can let the batch file decide which action to take next. In this article, we showed you how to set up the commands in a batch file that let your batch files make those decisions. ◆

COMMAND SHORTCUTS

VERSION 5.0 & 6.x

Putting the . and .. directory symbols to work

You've seen the . and .. symbols when you check the directory listing of a subdirectory. You probably know that a single period represents the current directory and a double period represents the current directory's parent directory. But, did you know that you can use the . and .. symbols as shortcuts in several DOS commands? In this article, we'll show you how. But first, let's look at a concept that's vital to understanding how to use these symbols: parent and child directories.

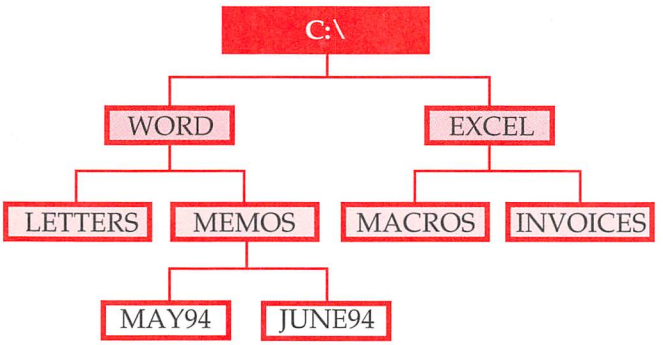
Parent and child directories

Understanding the concept of parent and child directories is essential to understanding the . and .. symbols. We put this concept in graphic form with the directory structure shown in Figure A.

As you can see, two directories branch off the C:\ root directory—WORD and EXCEL. These directories are called the root's *child directories*, since they branch off the root directory. The root directory is called the

parent directory of the WORD and EXCEL directories. Just as WORD and EXCEL are child directories of the root directory, LETTERS and MEMOS are child directories of the WORD directory. Conversely, WORD is the parent of the LETTERS and MEMOS directories.

Figure A



This directory structure shows the relationship between parent and child directories.

The directory hierarchy begins at the root directory, which is the only directory that doesn't have a parent. Below the root directory are directories that are children of the root directory and that may be parent directories themselves. Now let's look at how we normally enter commands that affect files in different DOS directories.

Navigating directories

Whenever you work at the DOS prompt, you're always working in the *current* directory. You should be aware of where you are in the directory structure—that's why we always recommend that you set your prompt to display the current directory path by using the command

```
PROMPT=$p$g
```

in your AUTOEXEC.BAT file. For example, if your DOS prompt reads

```
C:\EXCEL\MACROS>
```

you know that you're in a directory that's a grandchild of the root (C:\) directory—the MACROS directory's parent is EXCEL, and EXCEL's parent is C:\.

Normally, you'd change from the MACROS directory to the EXCEL directory by issuing the command

```
C:\EXCEL\MACROS>cd \EXCEL
```

You use the backslash to indicate that you're moving backward in the hierarchy, to a subdirectory of the root directory.

You use similar logic to issue a COPY command. If your DOS prompt reads

```
C:\EXCEL\MACROS>
```

and you want to copy all the files in the current directory back one level to the EXCEL directory, you'd use the command

```
C:\EXCEL\MACROS>copy *.* \EXCEL
```

You could construct these two examples as we've done in this section. However, you can save lots of keystrokes by using the . and .. symbols. DOS uses the symbol . to represent the current directory and the symbol .. to represent the current directory's parent directory.

Using the . and .. symbols

You can use the . symbol when you want a command to act on all the files in the current directory. You can use the .. symbol when you want a command to act on all the files in the current directory's parent directory. You can use . and .. as shortcuts when using three classes of DOS commands:

- Directory commands: DIR and CD
- Copy commands: COPY and XCOPY
- Delete commands: DEL and DELTREE (in DOS 6.x)

Let's explore using . and .. in each of these categories.

The directory commands

Since the CD command changes to another directory and since the DIR command defaults to the current directory, you wouldn't use the . symbol with these two commands. However, you can make use of the .. symbol with both directory commands.

You're probably already familiar with the most common use for the .. symbol—using CD to move up one level to the directory's parent. You simply use

```
C:\WORD\MEMOS>cd ..
C:\WORD>_
```

instead of the command

```
C:\WORD\MEMOS>cd \word
C:\WORD>_
```

If you want, you can chain together the .. symbol and a directory path to move up and then down through the directory structure. Try using the CD command with .. to move to a directory that's parallel to the directory you're in. For example, if you're in the C:\WORD\MEMOS\MAY94 directory, you could switch to C:\WORD\MEMOS\JUNE94 by entering two commands:

```
C:\WORD\MEMOS\MAY94>cd..
C:\WORD\MEMOS>cd june94
C:\WORD\MEMOS\JUNE94>_
```

However, you can make the switch much easier by using the single command

```
C:\WORD\MEMOS\MAY94>cd ..\june94
C:\WORD\MEMOS\JUNE94>_
```

You can use .. with the DIR command, as well. To view the directory of the current directory's parent, you simply use the command

```
C:\WORD\MEMOS\JUNE94>dir ..
```

instead of changing to the parent directory and entering *DIR* or using a command in the form

```
C:\WORD\MEMOS\JUNE94>dir \word\memos
```

The copy commands

The basic form of the XCOPY and COPY commands is

```
command source target
```

where *source* is the path and files you're copying and *target* is the path to which you're copying the files. You can use the copy commands to copy files between very remote directories or to disks in different drives. However, if you want to copy files from the current directory or to a parent directory, you can use the . and .. symbols to your advantage.

You can use . to copy all the files in the current directory to another directory, using the COPY or XCOPY command. For example, you can copy all the files from the EXCEL\MACROS subdirectory to the root directory by using the command

```
C:\EXCEL\MACROS>xcopy . c:\
```

instead of

```
C:\EXCEL\MACROS>xcopy *.* c:\
```

The .. symbol can save you keystrokes with the COPY and XCOPY commands, too. If you want to copy a file back one level, you can specify the .. symbol as the copy command's target directory. For example, to copy a file in the MACROS directory to its parent, the EXCEL directory, you use the command

```
C:\EXCEL\MACROS>copy bedinger.xlm ..
```

instead of the command

```
C:\EXCEL\MACROS>copy bedinger.xlm \excel
```

You can use both . and .. in a copy command, but only to copy all the files from the current directory to its parent directory. For example, you use the command

```
C:\EXCEL\MACROS>copy . ..
```

instead of

```
C:\EXCEL\MACROS>copy *.* \excel
```

to copy all the files from the MACROS subdirectory to the EXCEL directory.

The delete commands

The . and .. symbols can save you time when you want to delete all the files in the current directory or its parent directory, but you should be very careful when you use these symbols with the DEL and DELTREE commands.

You can use . with the DEL command to delete all the files in the current directory. For example, you can delete all the files in the WORD directory by using the command

```
C:\WORD\>del .
```

instead of the command

```
C:\WORD\del *.*
```

You can delete files in the parent directory using the .. symbol. For example, you can delete files in the MEMOS subdirectory from its child directory using the command

```
C:\WORD\MEMOS\MAY94>del ..
```

Be very careful using this command, especially if you're in a directory once removed from the root directory. It's far too easy to wipe out the system files in your C:\ directory by entering a *DEL ..* command from one of the C:\ subdirectories.

In DOS 6.x, you can use the DELTREE command with either . or .. to delete every file in the current directory and in any child directories of the current directory. DELTREE deletes hidden files, read-only files, and system files as well as visible files. For example, you can use the command

```
C:\WORD\MEMOS\MAY94>deltree .
```

or

```
C:\WORD\MEMOS\MAY94>deltree ..
```

to delete all the files and subdirectories in the MAY94 subdirectory. However, when you use DELTREE inside a directory, the command won't delete the current directory. You must move to the parent directory or another directory to remove the directory name.

Conclusion

DOS uses the . symbol to represent the current directory and the .. symbol to represent the current directory's parent directory. You can save a lot of keystrokes by using these symbols with commands you issue at the DOS prompt, as we showed in this article. ◆

VERSION 6.x

Adding an escape hatch for DOS 6's CHOICE command

Many of us breathed a sigh of relief when Microsoft added the CHOICE command to DOS in Version 6.0. At long last, we have a way to solicit user input in a batch file. You use the CHOICE command to list keys the user can press to execute a command or set of commands. The CHOICE command pauses the batch file until the user presses one of those keys. Then the batch file executes a command or set of commands depending on which key the user pressed. (We showed you how to use the CHOICE command in "An Easy Way to Make CHOICEs with a Batch File" in the August 1993 issue.)

Usually you list only keys that perform some action. The batch file waits until you tell it which action to execute. If you press any other key, the batch file beeps and continues to wait. In this situation, the only way to escape from the decision is to press [Ctrl]C—but that ends the batch file.

Sometimes you want to choose to *not* perform one of the specified actions. You want to act on the adage "Choosing not to choose is a choice in itself." In other words, you want an escape hatch. That's what we'll give you in this article. We'll show you how to add the [Escape] key as an option for the CHOICE command. First, let's look at how to add CHOICE to a batch file.

Constructing a CHOICE statement

You can use a number of switches with CHOICE, but the basic form of the command is

```
choice /c:option1option2option3...optionn text
```

where the /C switch indicates that you're specifying keys other than the default Yes/No. *Option1* through *option*n are the allowable keys, and *text* specifies text you want to display as a prompt. If your text message describes all the options, you'll want to add the /N switch to the CHOICE statement. The /N switch keeps DOS from displaying the choices and the ? character at the end of the text prompt. For example, you might add this CHOICE statement to a batch file

```
choice /n /c:AB Press A to copy files to drive A; B for
     drive B
```

to let the user decide which drive to copy files to.

Once you've defined the allowable keys, you add IF ERRORLEVEL statements. (We discuss this topic in detail in "Using ERRORLEVEL to Make Decisions in a Batch File" on page 6.) When you run the batch file, these statements determine which key the user presses. CHOICE assigns each of the keys you specify with the /C switch an ERRORLEVEL based on the order in which you listed the choices—the first choice, A, is ERRORLEVEL 1 and the second choice, B, is ERRORLEVEL 2. You enter the IF ERRORLEVEL statements in order from highest to lowest, for example:

```
if errorlevel 2 goto :COPYB
if errorlevel 1 goto :COPYA
```

The GOTO *label* commands send the batch file to the appropriate branch, depending on which key the user presses. In our example, you'd include in each labeled branch the commands that copy the files to a disk in the specified drive.

Now, let's broaden the scope of our sample batch file. Suppose the batch file's main purpose is to delete files you specify. Our CHOICE statement lets you copy the files to a floppy disk before you delete them from your hard disk. Now you want to add an option that lets you choose to delete the files without copying them first. We could use a letter for the third option, but let's add an option for the [Escape] key instead. We generally associate the [Escape] key with backing out of a certain action, so it's the perfect key for the job.

Adding an option for the [Escape] key

Adding the [Escape] key as an option for the CHOICE key is just as easy as adding any other key. All you need to know is the key's ASCII number and how to enter it into the CHOICE statement.

The [Escape] key is ASCII code 27, which is also known as control character ^[. In the DOS Editor, you add the [Escape] key as an option by pressing [Ctrl]P followed by [Escape]. (Alternatively, you can press [Ctrl]P followed by [Ctrl][, the open bracket symbol.) When you press this key combination, the Editor displays the ← symbol.

Let's add the [Escape] key option to our previous example. Reopen the batch file in the DOS Editor and place the cursor just beyond */c:AB*. Press [Ctrl]P followed by [Escape]. Press [End] to move the cursor to the end of the line and then add a note about the [Escape] option. Now the CHOICE statement looks like this:

```
choice /n /c:AB← Press A to copy files to drive A;
     B for drive B; or press [Escape] to delete
     specified files.
```

You next add an IF ERRORLEVEL statement to test for this new option. Since you added the option after the first two options, CHOICE will assign it an ERRORLEVEL of 3. Thus, you add the IF ERRORLEVEL statement before the other two in the batch file:

```
if errorlevel 3 goto :ESCAPED
if errorlevel 2 goto :COPYB
if errorlevel 1 goto :COPYA
```

Finally, you add a branch labeled :ESCAPED to your batch file. Of course, this branch should contain the commands that delete the specified files.

Conclusion

You can add more decision-making power to the CHOICE command by adding an option that allows the user to press the [Escape] key. In this article, we showed you how to add the [Escape] option to a CHOICE statement and how to modify your batch file to accommodate the new option. ◆

LETTERS

CHKDSK /F won't run in Windows

When I shell out to the DOS prompt from Windows and type in the command

```
chkdsk /f
```

to recover lost allocation units, I get the response

```
CHKDSK /F cannot be done in a Windows/DosShell Command
    Prompt
```

Can you tell me how to run the CHKDSK command to fix errors on the disk?

Leonard Sabella
Roslyn Heights, New York

DOS won't let you run CHKDSK /F from Windows or from the DOS Shell because those programs open files temporarily and then close or delete them when you exit the program. CHKDSK /F looks for lost clusters, which are spaces on the disk that contain information but have no entry in the file allocation table. Lost clusters usually occur when files are opened temporarily by an application but not closed or deleted through proper means. You can't run CHKDSK /F while Windows is running because the command will close any files your Windows applications are currently using (such as temporary files) and will update the file allocation table with incorrect information.

To use CHKDSK /F, you must first close your Windows applications and exit Windows. Then, run the command from the DOS prompt. Once you've recovered the lost allocation units, you can reopen Windows and continue with your work.

By the way, there are other commands you shouldn't run from inside Windows. Any command that can modify the file allocation table might not run properly from Windows. You should avoid using UNDELETE, APPEND, FASTOPEN, SCANDISK, SUBST, and disk-compaction and defragmentation programs, such as DOS's DEFRAG utility, from Windows. ◆

CHOICE command's ERRORLEVELs reported incorrectly

We gave some incorrect information in our April 1994 article "Creating a Safety Net for Your Batch Files." In the text, we stated that CHOICE assigns values to ERRORLEVEL beginning with the value 0 (zero). Actually, CHOICE assigns the first default option, Y, an ERRORLEVEL of 1; the second option, N, is 2; and so on. There should be only one IF ERRORLEVEL statement in Figure A on page 11:

```
if errorlevel 2 goto CANCEL
```

This statement will send the batch file to the :CANCEL section if you press *N* or let time expire. You don't need a second statement to send the batch file to :KILL if you press *Y*—it would be redundant, since the :KILL commands come next in the batch file. We're sorry for any inconvenience this caused, and we thank the readers who pointed out our error.